A logic of behaviour in context

Information and Computation ••• (••••) •••–•••

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

A logic of behaviour in context

C.J. Banks *, I. Stark

Laboratory for Foundations of Computer Science, School of Informatics, The University of Edinburgh, 10 Crichton St., Edinburgh EH8 9AB, United Kingdom

ARTICLE INFO

Article history:
Available online xxxx

ABSTRACT

We present a novel temporal logic for expressing properties of *behaviour in context*. The logic is applied to models of continuous processes, specifically using the continuous π-calculus as a modelling language for biochemical systems.

The logic allows the expression of the temporal behaviour of a system when placed in the context of another system. Here we study this in terms of biochemical reactions and the expression of temporal behaviour in the context of other biochemical processes. We present the syntax and semantics of the logic and study the model-checking problem over continuous time and continuous state-space process models, using the continuous π-calculus.

We present a succinct, but naive, model-checking algorithm and then show how this can be improved. We investigate the complexity of model-checking, where repeated ODE solving emerges as a particular cost; assess some limitations of the technique; and identify potential routes to overcome these.

1. Introduction

In this paper we present a novel temporal logic for expressing properties of *behaviour in context*. That is, given a model, we may express the temporal behaviour of the model when composed with some other model. Put another way, one can express properties of the behaviour of a system given the introduction of some external influence.

The logic, $\mathcal{LBC}$, is based on the classical Linear Temporal Logic. $\mathcal{LBC}$ introduces the *context modality*, $C \triangleright \phi$, where C is some context and ϕ, the behaviour, is a formula. This modality is inspired by the guarantee operator in the spatial logic of Cardelli and Gordon [1], but with reduced expressive power to allow a tractable implementation.

We study this logic as applied to expressing the properties of biochemical processes. For example, given some process model P, assume the context C is to be some new process Q; then the assertion $P \models Q \triangleright \phi$ expresses that P, when composed with Q, has the behaviour ϕ.

On the surface it appears that the context modality adds little that cannot be expressed by first composing models then making an assertion in classical temporal logic. However, it is when the context modality is nested within the temporal modalities that we see the gain in expressive power. For example we can make assertions like *if we introduce process Q into the model at any point in the future then some behaviour ϕ will be observed*, that is $\mathbf{G}(Q \triangleright \phi)$ in $\mathcal{LBC}$.

To give a model over which to interpret $\mathcal{LBC}$ we use the continuous π-calculus (cπ) to model biochemical processes. cπ is a continuous time and continuous state-space process algebra, an expressive and succinct language for modelling the

* Corresponding author.
E-mail addresses: C.Banks@ed.ac.uk (C.J. Banks), Ian.Stark@ed.ac.uk (I. Stark).
URLs: http://banks.ac/ (C.J. Banks), http://homepages.ed.ac.uk/stark (I. Stark).

0890-5401/$ – see front matter © 2014 Published by Elsevier Inc.
http://dx.doi.org/10.1016/j.ic.2014.01.009

behaviour of biochemical processes. In Section 2 we describe the necessary details of cπ. Then, in Section 3, we present the full syntax and semantics of $\mathcal{LBC}$.

In order to define a model-checking algorithm for $\mathcal{LBC}$ over cπ processes we follow the example of Antoniotti et al. [2] and Calzone et al. [3] to approximate model-checking of a continuous state-space by using a discrete simulation trace of the state-space. In this case the ordinary differential equations representing the state-space are solved numerically and model-checking is done over a finite numerical trace. In Section 4 we discuss a naive algorithm which demonstrates the model-checking procedure.

The computation time of the naive algorithm grows exponentially with the depth of nested temporal modalities. To address this we draw from the dynamic programming style algorithm of Calzone et al. [3]. However, the dynamic programming algorithm suffers from the lack of short-circuiting. That is, in many cases is it not necessary to check the entire length of the simulation trace to decide the truth of a formula, but the dynamic programming algorithm requires a full traversal of the trace.

We then define a hybrid algorithm, which whilst employing a dynamic programming approach, allows short-circuiting. This turns out to run faster in practice than the other algorithms.

We show that the computation time of model-checking is exponential in the *sandwich alternation depth* of the formula, for all these algorithms. This is a variant on classic alternation: time complexity is exponential in the depth of nested temporal modalities where they are separated by context modalities.

Furthermore, a major constant in the computation time of this technique is the cost of calling the numerical solver. Calls to the solver greatly outweigh the cost of traversing the trace. The solver needs to be called in order to evaluate a context modality and possibly once for every time point in the trace when a context modality is nested below a temporal modality. We will describe some methods for reducing the number of calls which need to be made to the solver. This is also a focus of future work. Section 5 sums up these conclusions and gives more details of further work.

2. Continuous π-calculus

The continuous π-calculus (cπ) was designed as a formal language for the study of evolutionary variation in biochemical processes. The canonical reference for cπ is Kwiatkowski's thesis [4], but the original language semantics were first published by Kwiatkowski and Stark [5].

The language syntax is based on the π-calculus of Milner [6], with some alterations and additions to better support the description of biochemical models. The usefulness of π-calculus style languages for modelling biochemical processes is well established, having first been described by Regev et al. [7,8].

The description of a biochemical process in cπ is split into two levels: *species* and *process*. A *species* in cπ is a description of the behaviour and interaction capability of a biochemical species. This level is similar to a π-calculus process. A *process* in cπ is a real-indexed parallel composition of each of the species in the biochemical process, representing a mixture with some initial concentration of each species.

In cπ each species involved in a process is represented by a term which describes its possible behaviours. For example, if we consider a simple chemical reaction:

$$A + B \to C + D$$

then we can define A as:

$$A \triangleq a.C$$

That is we have a type of molecule A which has a reaction site a. It can react on a and becomes C. Then we can define B as:

$$B \triangleq b.D$$

That is we have a type of molecule B which has a reaction site b. It reacts on b and becomes D. Below we will define how A and B interact. Now let us say that C degrades over time (possibly by some external process which we do not wish to describe, or wish to abstract away from):

$$C = \tau_d.0$$

Here C has no reaction sites. All it does is the special (autonomous) τ action at rate d and becomes 0 (nil) which is the inert species, it has no further action. Then let us say that D is an inert species, it has no action and just accumulates in our system:

$$D = 0$$

it behaves as the nil species.

Instead of the prescribed co-naming scheme of the π-calculus, cπ uses *affinity networks* to support arbitrary interaction. The affinity network is an undirected weighted graph of the sites with arcs recording the potential to react and the weight recording the reaction rate; thus the affinity for reaction between any biochemical species can be easily defined. Local

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

affinity can be assigned dynamically in the case, for instance, of complexation where there is a local interaction between a bound substrate and enzyme (see example in Section 2.3).

For our simple reaction the global affinity network N will allow sites a and b react at rate k:

$$N = \{a \overset{k}{-} b\}$$

To complete our model we then give a process term, defining our process Π as a composition of its species and their initial concentrations:

$$\Pi \triangleq 1.0 \cdot A \parallel 1.0 \cdot B \parallel 0.0 \cdot C \parallel 0.0 \cdot D$$

where we initially have a 1.0 Molar concentration of each A and B, and no C or D is present.

Each cπ process evolves continuously over time through a real-valued state-space. In the example with four species we have a four-dimensional space, where any point in this space represents a possible state of the system. The behaviour of our system is a trajectory through this space. The trajectory for this example begins at the initial values $(1.0, 1.0, 0.0, 0.0)$, the concentrations of A and B will decrease at the same rate to zero, the concentration of C will increase initially and then decrease to zero as it degrades, and D will increase until A and B run out.

The remainder of this section gives the precise syntax and semantics of cπ as well as a more substantial example.

2.1. Syntax

The syntax of cπ is given formally by the following set of definitions.

Definition 1 *(Name).* A *name* (a, b, x, y, etc.) is a member of a countably infinite, totally ordered, set of names $\mathcal{N}$. The notation $\vec{a}, \vec{b}, \vec{x}, \vec{y}$, etc. is used to denote finite vectors of names.

Definition 2 *(Prefix).* A *prefix* is a *communication prefix* of the form $a(\vec{x}; \vec{y})$ or a *silent prefix* of the form τ_k where $k \in \mathbb{R}$. The notation π, π', π_i, etc. is used to denote a prefix.

A communication prefix $a(\vec{x}; \vec{y})$ denotes potential for reaction on the channel (site) a. Names in $\vec{x}$ are sent in the reaction and names in $\vec{y}$ are received. In a term $a(\vec{x}; \vec{y}).A$ names in $\vec{y}$ are binding with scope A. An occurrence of a name in a term is bound if it is, or it lies in the scope of, a binding occurrence of the name. A name is free if it is not bound (see Definition 6).

Definition 3 *(Affinity network).* An *affinity network* is a finite undirected weighted graph whose vertices are names and whose edges are weighted with $k \in \mathbb{R}$. The notation M, N, K, etc. is used to denote affinity networks.

Definition 4 *(Species).* The set $\mathcal{S}$ of *species* is defined by the following grammar:

$$\begin{aligned} \text{(Species)}\ A, B &::= 0 \mid \sum_{i=0}^{n} \pi_i.S_i \mid A|B \mid (\nu M)A \\ \pi &::= a(\vec{x}; \vec{y}) \mid \tau_k \\ S &::= D(\vec{a}) \mid A \end{aligned}$$

where:

- The *nil* species 0 denotes a species incapable of any action.
- A definition $D(\vec{a}) \triangleq A$, where $\vec{a}$ is a vector of the free names of A, defines a species constant. The invocation $D(\vec{a})$ behaves as the body A with $\vec{a}$ substituted for the free names of A.
- A prefix $a(\vec{x}; \vec{y})$ may be denoted $a(\vec{y})$ when $|\vec{x}| = 0$, $a\langle\vec{x}\rangle$ when $|\vec{y}| = 0$, or a when $|\vec{x}| = |\vec{y}| = 0$.
- The *choice* $\sum_{i=0}^{n} \pi_i.S_i$ denotes a mutually exclusive choice of interaction; upon performing the interaction π_i the resultant species behaves as A_i. For binary choice the shorthand notation $\pi_1.A_1 + \pi_2.A_2$ may be used.
- The *species composition* $A|B$ denotes a complex of A with B.
- *Name restriction* $(\nu M)A$ restricts the scope of the names in M to be local in the species A. Restriction itself has no direct biochemical correspondence, but is used as a mechanism for defining local names which denote the interactions between components of a complex.

Definition 5 *(Process).* The set $\mathcal{P}$ of *processes* is defined by the following grammar:

$$\text{(Process)}\ P, Q ::= c \cdot S \mid P \parallel Q$$

A *process* may be a species with an initial concentration $c \in \mathbb{R}$, or a process composition (mixture) of species.

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

Definition 6 *(Free names).* The free names of a species are defined by the function $\mathtt{fn} : \mathcal{S} \to \mathcal{N}$, defined recursively as:

$$\begin{aligned} \mathtt{fn}(0) &= \emptyset & \mathtt{fn}(D(\vec{a})) &= \vec{a} \\ \mathtt{fn}(A|B) &= \mathtt{fn}(A) \cup \mathtt{fn}(B) & \mathtt{fn}((\nu M)A) &= \mathtt{fn}(A) \backslash M \\ \mathtt{fn}(\tau_k.A) &= \mathtt{fn}(A) & \mathtt{fn}(a(\vec{x};\vec{y}).A) &= \{a\} \cup \vec{x} \cup (\mathtt{fn}(A) \backslash \vec{y}) \\ \mathtt{fn}(\textstyle\sum_{i=0}^{n} \pi_i.A_i) &= \textstyle\bigcup_i \mathtt{fn}(\pi_i.A_i) \end{aligned}$$

If X, Y are sets of names and A is a species, then we denote by $X\#A$ that X *is fresh for* A; that is $X \cap \mathtt{fn}(A) = \emptyset$. Similarly, $X\#Y$ denotes $X \cap Y = \emptyset$.

Definition 7 *(Structural congruence).* A structural congruence $\equiv$ is defined over both species and processes, equating syntactically congruent objects:

$$\begin{aligned} 0|A &\equiv A & & & (c \cdot 0) \parallel P &\equiv P \\ A|B &\equiv B|A & & & P \parallel Q &\equiv Q \parallel P \\ (A|B)|C &\equiv A|(B|C) & & & (P \parallel Q) \parallel R &\equiv P \parallel (Q \parallel R) \\ \Sigma_{i=0}^{n} \pi_i.A_i &\equiv \Sigma_{i=0}^{n} \pi_{\sigma_i}.A_{\sigma_i} & \text{perm. } \sigma & & (c+d) \cdot A &\equiv (c \cdot A) \parallel (d \cdot A) \\ (\nu M)A &\equiv A & M\#A & & c \cdot (A|B) &\equiv (c \cdot A) \parallel (c \cdot B) \\ (\nu M)(\nu N)A &\equiv (\nu N)(\nu M)A & M\#N & & c \cdot A &\equiv c \cdot B \qquad A \equiv B \\ (\nu M)(A|B) &\equiv A|(\nu M)B & M\#A & & & \end{aligned}$$

2.2. Semantics

It is beyond the scope of this paper to formally define the cπ semantics here (see Kwiatkowski's thesis [4] for a full formal definition). Indeed it is not necessary, as long as the reader understands that a cπ process generates a set of ordinary differential equations which describe the behaviour of the system. However, we will describe the semantics briefly and informally.

The meaning of a cπ model is split into two levels, in the same way as the syntax. The set of species is given a discrete transition system which gives its interaction capabilities. The process is then given a continuous-time continuous-space semantics, giving the dynamic behaviour of the process. The output of this is a set of coupled ordinary differential equations describing the dynamics, but the calculus itself has a compositional internal representation of the dynamics.

The species transition system is given by a set of rules governing operational semantics in a standard process calculus manner, using the Structural Operational Semantics style. The transition system is a multi-set of transitions in this case, because multiple capabilities for the same behaviour need to be accounted for in a quantitative setting.

The established approach (e.g. in BioCHAM [9] and PEPA [10,11]) to giving a continuous-time continuous-space semantics to processes is to directly translate the syntactic objects into equivalent Ordinary Differential Equations (ODEs) and a set of initial values. Whilst the process languages themselves are compositional, ODEs are not compositional — it is not possible to derive the ODEs governing the behaviour of the whole process from those of its sub-processes. In cπ a process is given a compositional vector-space semantics in the following way.

The behaviour of a cπ process is given by $\frac{dP}{dt} \in \mathbb{P}$ where:

- the *process space* $\mathbb{P}$ is the vector space $\mathbb{R}^{(\mathcal{S}^{\#})}$, where $\mathcal{S}^{\#}$ is the set of *prime species* — elementary species which cannot be broken down into a composition of two non-nil species, modulo $\equiv$;
- the gradient vector $\frac{dP}{dt}$ specifies a trajectory through the process space.

$\mathbb{P}$ is the phase space of a cπ model and the dynamic behaviour is a trajectory through this space starting from the initial state (concentrations). This is equivalent to a set of ODEs governing the whole behaviour.

To arrive at a compositional model some extra information is needed; that is the potential interactions. The immediate behaviour of a composition of processes is simply the sum of the immediate behaviours of the component processes and the behaviour that emerges from their interaction, given the potential interactions they can undertake.

Thus a compositional model is obtained. This aspect of the formalism is described in detail in Kwiatkowski's thesis [4]. Essentially, a cπ model has a semantics in terms of phase portraits; these phase portraits are computed from those of its sub-systems and can be composed.

For analysis, a set of ODEs representing each species in the system can be derived by means of the extraction algorithm in Kwiatkowski's thesis.

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

2.3. Example

The following is a slightly more complicated example of modelling in cπ than our first example. It shows how enzyme-catalysed protein–protein interactions can be modelled in cπ. This concept can be used as a basis for building more complex networks of interacting proteins. It uses all the cπ syntax and provides an intuition for the semantics of processes.

Consider the Michaelis–Menten reaction:

$$S + E \rightleftarrows ES \rightarrow E + P$$

where a substrate S binds to an enzyme E to form a complex ES. The complex can either unbind and release the substrate and enzyme, or it can react and release a product P and the unaltered enzyme. In cπ we can model this in the following way.

First we will define our simplest species, the product:

$$P \triangleq \tau_{r_{deg}}.0$$

In this case our product degrades at some rate r_{deg} and does nothing else; it becomes the inert 0 species. This is modelled as an autonomous τ reaction.

Next we model the substrate:

$$S \triangleq s(u, r).(u.S + r.P)$$

The substrate has a channel s; in cπ a channel represents a reaction site. After reacting on the site s, it has a choice of interactions: it can react on u to become the substrate again, or it can react on r to become the product. The channels u and r are received when something reacts on s.

Our final species is the enzyme:

$$E \triangleq \left(\nu \left\{ {}_t\angle^u_r \right\}\right) e\langle u, r\rangle.t.E$$

The enzyme has a channel e. After reacting on e it can only do one thing: react on t and become the enzyme in its initial state again. When reacting on e the enzyme sends out two channels u and r. The channels u, r, and t are local to the enzyme; they are bound in the ν binder, which defines a local affinity graph which allows u and t to react at rate r_u or r and t to react at rate r_r.

If we now imagine that we have defined a global affinity graph that states that channels s and e can react, then we are stating that these two sites can react. Upon reaction, the local channels u and r are sent on e and received on s and we form the complex:

$$ES \equiv \left(\nu \left\{ {}_t\angle^u_r \right\}\right)\left(t.E \mid (u.S + r.P)\right)$$

The complex now carries the local affinity graph from the enzyme and is a composition of the term $t.E$ from the enzyme and the term $u.S + r.P$ from the substrate. As all channels in the complex term are bound by the local affinity graph, it has no external reaction capability. The complex can only perform one of two internal reactions: t and u can react to give E and S — the unbinding — or t and r can react to give E and P — the formation of the product and release of the enzyme. Note that the term for the complex does not need to be defined, it emerges as the result of the binding reaction.

Finally, we define the process term, which lists the species in our initial mixture and their initial concentrations $c_{i\in\mathcal{S}}$:

$$\Pi \triangleq c_S \cdot S \parallel c_E \cdot E \parallel c_P \cdot P$$

and the global affinity graph which allows s and e to react:

$$N = \{s \overset{r_b}{-} e\}$$

Note that the arcs in the affinity graphs will be labelled with the reaction rates: r_b for the binding rate of E and S, r_u for the unbinding rate, and r_r for the rate at which the complex forms product.

The model can now be compiled and a set of ODEs extracted using the ODE extraction algorithm. This results in one equation per species, both the initially defined species and any that arise from reaction — in this case, the complex:

$$[E]' = r_r[ES] + r_u[ES] - r_b[E][S]$$

$$[S]' = r_u[ES] - r_b[E][S]$$

$$[P]' = r_r[ES] - r_{deg}[P]$$

$$[ES]' = r_b[E][S] - r_u[ES] - r_r[ES]$$

where $[A]$ is the concentration of A and $[A]'$ is the first derivative of $[A]$. The ODEs can be solved numerically, given values for the rates and initial concentrations, and we produce the time series in Fig. 1.

The above model is essentially an abstract template for any enzyme catalysed reaction. For an example from biology, if we consider part of a MAPK signalling pathway [12], the protein kinase *Ras* promotes the phosphorylation of *Raf* to *Raf**. To model this we simply substitute *Raf* for S, *Ras* for E, and *Raf** for P in the above model; then we may substitute the rate parameters for those taken from a biological database.

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

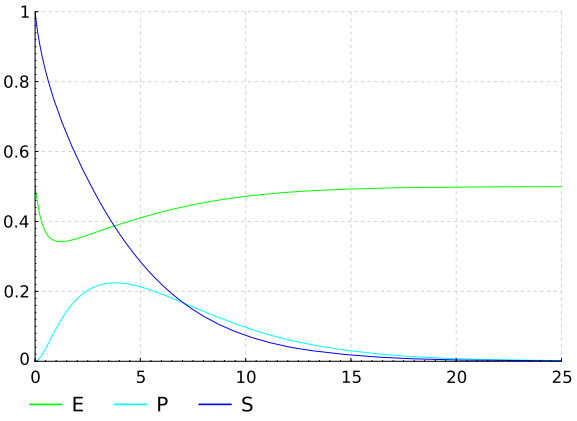

Fig. 1. Plot of the time series of the Michaelis–Menten reaction model.

2.4. Larger models

It is easy to take small models of interacting proteins, like the one above, and from them build larger models of interacting protein networks. The compositionality of the language reduces the work needed to combine components, compared to combining ODE models which are inherently non-compositional.

For example, our small model for phosphorylation of *Raf*, above, can be combined with small models of the other components of a MAPK cascade to form a larger model of the cascade overall.

A number of non-trivial $c\pi$ models have been built. Examples include the MAPK signalling pathway and Kai circadian clock in Kwiakowski's thesis [4]. The Kai circadian clock was also re-modelled using the more recent tools and some analysis performed in Clark et al. [13]. Models have also been built of the idealised posttranslational biochemical oscillator of Jolley et al. [14] and the molecular titration models of Buchler and Louis [15]. These models demonstrate that $c\pi$ can deal efficiently with models whose behaviour is complex, e.g. oscillatory.

3. Logic of behaviour in context

$\mathcal{LBC}$ arose from the desire to define a logic for $c\pi$ which would allow the classification of the behaviour of a $c\pi$ model. It was clear that a temporal logic, and a logic which allowed the expression of constraints on the real-valued concentrations of $c\pi$ species, was required.

However, particularly in biochemical systems, behaviour is often reasoned about in terms of, not just the system itself, but the system's behaviour when it is perturbed somehow. We wish to be able to reason about the system's behaviour in some external context. Thus $\mathcal{LBC}$ was conceived.

$\mathcal{LBC}$ combines LTL($\mathbb{R}$) (see Calzone et al. [3]) and MITL (see Alur et al. [16]) with the addition of a *context modality*. LTL($\mathbb{R}$) gives atomic propositions with inequalities and arithmetic operators ranging over the real-valued concentrations of species. MITL gives the power to express real-time properties.

A context modality $Q \triangleright \psi$ holds for a process where ψ holds in the presence of a new process Q. This allows the expression of behaviour in some given context.

Using the context modality allows assertions such as:

$$c \cdot In \triangleright \mathbf{G}\big([Pr] < x\big)$$

In a biochemical context this could mean: in the presence of a concentration c of an inhibitor In the concentration of product Pr in the process always remains below x.

The context modality is based on the guarantee operator in the spatial logic of Cardelli and Gordon [1]. However, the guarantee took a logical formula on the left hand side $\phi \triangleright \psi$ meaning that the formula held for a process satisfying ψ in the presence of a process satisfying ϕ, that is:

$$P \models \phi \triangleright \psi \quad \Longleftrightarrow \quad (\forall Q \,.\, Q \models \phi \Longrightarrow P \parallel Q \models \psi)$$

Model-checking a logic with this guarantee is hard because of the necessity to quantify over all processes that satisfy an arbitrary formula. Caires and Lozes [17] give an account of the undecidabilty of spatial logic with the guarantee.

The context modality, however, gives some of the power of guarantee in a more tractably implementable manner by reducing the left hand side to a specific process:

$$P \models Q \triangleright \psi \quad \Longleftrightarrow \quad Q \parallel P \models \psi$$

A similar approach is taken by de Nicola and Loreti [18] who define a logic MoMo with a production operator which is based on guarantee. However, MoMo is defined for mobile processes with resources and locations, modelled using a

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

formalism based on shared tuple spaces, and the semantics of the production operator is incompatible with the type of continuous state processes we discuss here.

3.1. Syntax

Definition 8 *($\mathcal{LBC}$ formula).* The syntax of $\mathcal{LBC}$ formulae ϕ, ψ is defined by the following grammar:

$$\begin{aligned}
\phi, \psi &::= Atom \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \implies \psi \mid \neg\phi \\
&\quad \mid \phi \mathbf{U}_I \psi \mid \mathbf{F}_I \phi \mid \mathbf{G}_I \phi \mid Q \triangleright \phi \\
Atom &::= \top \mid \bot \mid Val\ ROp\ Val \\
Val &::= v \in \mathbb{R} \mid [A] \mid Val\ AOp\ Val \\
ROp &::= > \mid < \mid \geqslant \mid \leqslant \\
AOp &::= + \mid - \mid \times \mid \div
\end{aligned}$$

where Q is a process and I is a (time) interval in $\mathbb{R}_{\geqslant 0}$. We use the abbreviations $\mathbf{U}$, $\mathbf{F}$, and $\mathbf{G}$ to denote $\mathbf{U}_{[0,\infty)}$, $\mathbf{F}_{[0,\infty)}$, and $\mathbf{G}_{[0,\infty)}$ respectively and $\mathbf{U}_t$, $\mathbf{F}_t$, and $\mathbf{G}_t$ to denote $\mathbf{U}_{[0,t)}$, $\mathbf{F}_{[0,t)}$, and $\mathbf{G}_{[0,t)}$ respectively. $\top$ and $\bot$ denote *true* and *false* respectively. $[A]$ denotes the concentration of a species. Relational operators *ROp* and arithmetic operators *AOp* have the usual meaning.

3.2. Examples

In order to give some idea of the expressiveness of the logic and an intuition for the semantics, we present some simple motivating examples. These examples serve to give the reader an idea of the types of properties which can be expressed by $\mathcal{LBC}$. We begin with some examples of the basic temporal formulae.

The basic temporal modalities are fairly easy to understand: **F** is *future* (eventually) and **G** is *globally* (always). $\mathbf{F}([A] \geqslant c)$ means eventually the concentration of A will be at least c. $\mathbf{G}([A] > 0)$ means the concentration of A will always be greater than zero, i.e. we will never run out of A.

Although **F** and **G** correspond to the classical temporal logic modalities, they can be defined in terms of the **U** modality. The **U** modality can be useful in its own right for expressing changing conditions. $([A] > 5)\mathbf{U}([B] > 1)$ means the concentration of A is greater than 5 *until* the concentration of B is greater than 1, moreover the concentration of B will eventually be greater than 1.

If we specify time bounds on the modalities then we can be more precise about when something is true:

- $\mathbf{F}_{24}([A] > 0)$ means *at some point up to time* 24 the concentration of A will be greater than zero,
- $\mathbf{G}_{[10,15)}([A] \leqslant 1)$ means *between times* 10 *and* 15 the concentration of A is always 1 or less.

and combining temporal modalities allows us to encode ever more complex properties, e.g.:

- $\mathbf{F}(\mathbf{G}([A] > c))$ means eventually the concentration of A will be greater than c and will remain so indefinitely.

Now we present some examples of the use of the context modality. By combining the context modality with the temporal modalities we can formulate precise statements about the behaviour of a system when we introduce something new to it. At the top level this is a fairly simple expression of a change of initial conditions:

- $Q \triangleright \mathbf{F}([A] \geqslant c)$ means in the presence of Q eventually the concentration of A will be at least c.
- $(Q \triangleright \mathbf{F}([A] \geqslant c)) \wedge \neg\mathbf{F}([A] \geqslant c)$ is a stronger statement; not only do we state that the above property is true, but that indeed it is not true when Q is not present. Therefore we can say that it is Q which causes the concentration of A to rise to at least c.

However, if we begin to nest the context modality below the temporal modalities then we can express more complex properties about the times at which something is introduced:

- $\mathbf{G}(Q \triangleright \mathbf{F}([A] < c))$ means that if we introduce Q *at any time* in the process then eventually $[A] < c$,
- $\mathbf{F}(Q \triangleright \mathbf{F}([A] < c))$ means that *there exists some time point* in the process such that if Q is introduced then eventually $[A] < c$,

and, again, including time bounded modalities allows one to be more precise about the time at which something is introduced:

- $\mathbf{G}_{[0,5)}(Q \triangleright \mathbf{F}([A] < c))$ means if we introduce Q *at any time between* 0 *and* 5 then eventually $[A] < c$.

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

3.3. Temporal experiments in biology

A good example, from a biological perspective, of a type of model which would particularly benefit from this sort of expressiveness is one with oscillatory behaviour. Typical models which exhibit this sort of behaviour are the Kai circadian clock [13] and Jolley et al.'s posttranslational oscillator [14].

Suppose we have some species S in the model, the concentration of which oscillates under normal conditions, and a temporal property $Osc([S])$ which states that $[S]$ oscillates. We could then formulate a simple experiment by checking $R \triangleright Osc([S])$. This will be true if upon the introduction of another species R to the model $[S]$ still oscillates.

However, much more complex experiments could be expressed by using the context modality within a temporal modality. For example we could check, not only that $[S]$ still oscillates with the introduction of R, but that no matter at what point in the oscillation cycle we introduce R then $[S]$ still oscillates: $\mathbf{G}(R \triangleright Osc([S]))$. Likewise, we might state a property $\mathbf{F}_{[m,n)}(R \triangleright \neg Osc([S]))$ where $[m, n)$ is a specific interval of interest in the oscillation cycle. If this property is true then there exists a point in this interval where the introduction of R kills the oscillation.

One could imagine taking this idea of formulating experiments further and, for example, using some parameter fitting technique to find a precise interval $[m, n)$ for which our formula holds; thereby one could find a precise interval in the cycle where the introduction of R kills the oscillation.

It could be considered that the kind of discrete event simulation implemented by existing software packages such as COPASI [19] is an instance of the operational capability of the context modality. However, as the above examples should serve to show, $\mathcal{LBC}$ formulae can express much richer properties than simple discrete events by embedding these in temporal logic. Moreover, $\mathcal{LBC}$ formulae represent a succinct and formal specification of behaviour in a given context, rather than just an operation on a model.

3.4. Semantics

Here we define the semantics of $\mathcal{LBC}$ formulae by giving the satisfaction relation $\models$ over $\mathcal{LBC}$ formulae and cπ processes. A formula ϕ is true of a process P if and only if $P \models \phi$.

Definition 9 *($\mathcal{LBC}$ satisfaction relation).* For $P \in \mathcal{P}$, a cπ process, and $\mathcal{LBC}$ formulae ϕ and ψ the satisfaction relation $\models$ is defined inductively as follows:

$$
\begin{aligned}
P &\models Atom && \iff && Atom \text{ is true in the initial state of } P \\
P &\models \phi \wedge \psi && \iff && P \models \phi \text{ and } P \models \psi \\
P &\models \neg\phi && \iff && P \not\models \phi \\
P &\models \phi \mathbf{U}_I \psi && \iff && \text{for some } t \in I,\ P^t \models \psi \text{ and for all } t' \in [0, t],\ P^{t'} \models \phi \\
P &\models Q \triangleright \phi && \iff && (Q \parallel P) \models \phi
\end{aligned}
$$

where Q is a cπ process with any new global affinity network. Process P^t is the state reached from process P after time t, that is, the initial concentrations of P^t will be the component species concentrations after P has evolved for time t. The notation P^t is shorthand for a function $\mathcal{P} \times \mathbb{R}_{\geqslant 0} \to \mathcal{P}$.

The remaining propositional connectives can be derived in the normal way and the remaining temporal modalities can be defined as follows:

$$\mathbf{F}_I \phi \equiv \top \mathbf{U}_I \phi$$

$$\mathbf{G}_I \phi \equiv \neg \mathbf{F}_I \neg \phi$$

Although here we define the semantics of $\mathcal{LBC}$ formulae over cπ processes, it is of course possible to generalise the definition of the logic to any compositional process model.

4. Model-checking

To check a cπ model against a specification in $\mathcal{LBC}$ we must consider the following problem: we cannot algorithmically check the whole infinite state-space. The state of the model is changing continuously over dense time. Therefore we can only reasonably consider a compact subset of the state-space.

Our method here for model-checking a continuous process is based on that of Calzone et al. [3] following the ideas of Antoniotti et al. [2]. First it is necessary to numerically solve the ODEs for the process model and obtain a discrete trace of the evolution of the system. A function $solve : \mathcal{P} \to Trace$ is assumed, where a numerical ODE solver gives us a *Trace* of the form:

$$(t_0, \vec{c_0}), (t_1, \vec{c_1}), \ldots, (t_n, \vec{c_n})$$

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

where $t_i \in \mathbb{R}$ is a discrete time point and $\vec{c}_i$ is a vector of the concentrations of each species in the process at that time point. In this way we are checking over a finite and compact state-space. Some assumptions must be made about the discretisation of the state-space and these will be discussed in Section 4.1.1.

The remainder of this section will consider a sequence of algorithms. We begin with a simple and intuitive algorithm which captures the semantics, but has poor computational complexity. We then show how to improve the algorithm to allow for a more tractable implementation.

4.1. Naive algorithm

A minimal, naive algorithm for model-checking $\mathcal{LBC}$ over traces of this form is defined quite succinctly by the following functional pseudocode:

```
check :: Trace → Formula → Bool

check (t:ts) (atom)      =  valid atom t
check (t:ts) (φ ∧ ψ)     =  (check (t:ts) φ) AND (check (t:ts) ψ)
check (t:ts) (φ ∨ ψ)     =  (check (t:ts) φ) OR (check (t:ts) ψ)
check (t:ts) (¬φ)        =  NOT(check (t:ts) φ)
check (t:ts) (φ U_[t0,tn] ψ)  =  if (time(t) < t0) then check ts φ U_[t0,tn] ψ
                                 else (time(t) ⩽ tn) AND ((check (t:ts) ψ) OR
                                 ((check (t:ts) φ) AND (check (ts) (φ U_[t0,tn] ψ))))
check (t:ts) (Q ▷ φ)     =  check (solve (compose Q (proc t)))) φ

solve :: Process → Trace
compose :: Process → Process → Process
proc :: State → Process
```

The base case for atomic propositions calls a function `valid` which simply checks that the constraints on species concentrations is satisfied at the current time point in the trace. For example, with $[A] \geqslant 0.1$ it checks the value of c_A corresponding to time point t is greater than or equal to 0.1.

Checking a context modality involves taking the current process (`proc`), composing it with the introduced process (`compose`), computing the trace for this new process (`solve`), and checking the introduction's sub-formula over this new trace.

4.1.1. Assumptions

For this technique of model-checking over traces — and therefore each of the algorithms presented in this paper — it is assumed that the trace is of sufficient length (in time) to allow verification of the formula. This assumption can be made because the formula itself specifies the time interval it needs to check. Notice, though, that it is not always possible to verify a formula with an interval which is not right-closed, e.g. $\mathbf{G}_{[0,\infty)}\phi$ is unsatisfiable using this method.

Another assumption which must be made is that the number of time points in the trace, especially where the derivative changes abruptly, is sufficient to represent an accurate approximation of the ideal model. In particular, the maximum distance between time points must be less than the minimum diameter of any interval in the temporal modalities. For example: consider a trace with time points at times $\{1, 4, 7, 10\}$ and a formula $\mathbf{F}_{[5,6)}(\top)$. $\mathbf{F}(\top)$ is certainly always *true*, but using the trace method we would check time point 4, find it is less than the lower bound 5, then check time point 7 and find it is greater than the upper bound 6, causing the result to incorrectly evaluate to *false*.

The latter assumption is a harder assumption to make. In practice, if a suitably large number of time points is requested from an adaptive step-size solver then we can have reasonable confidence in the result. However, there are no guarantees and this is a limitation of the technique. This said, for non-critical applications it is reasonable to assume the numerical solver gives us a good approximation of the real system and well tested means of analysing dynamical systems. Indeed, the work of Calzone et al. relied on the same assumptions.

In Section 5.1 we outline some future work which hopes to eliminate the need for some of these assumptions.

4.1.2. Complexity

The worst case time complexity for this naive algorithm is exponential in the size of formula. That is $O(n^f)$ where n is the length of trace and f is the depth of nested temporal ($\mathbf{U}$) formulae. This is owing to how the recursion unfolds for nested $\mathbf{U}$ formulae. This is true for just the temporal fragment of the logic and therefore for checking the full logic $\mathcal{LBC}$.

If we consider $\mathbf{G}(\mathbf{F}\phi)$ where ϕ does not become true until the end of the trace then we see that the trace is traversed n times, $\mathbf{F}\phi$ being re-checked unnecessarily at each time point. In the next section we present an algorithm which eliminates this unnecessarily repeated computation.

Note that in practice the most significant cost is that incurred by solving the differential equations. We will discuss this further in Section 4.4.

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

4.2. Dynamic programming algorithm

The dynamic programming algorithm used by Calzone et al. [3] is an established method for model-checking temporal logic over traces and a partial answer to this problem. By traversing the trace only once, checking each sub-formula of the formula at each time point, we avoid unnecessary re-traversals of the trace.

An algorithm using this technique proceeds as follows:

1. By post-order depth-first traversal of the formula we obtain sub-formulae ordered by dependency.
2. Reverse the ordering of the trace.
3. We traverse the reversed trace once, labelling each time-point with each sub-formula in order if it holds, according to the following rules:
   - An atomic proposition holds if its constraint is satisfied.
   - $\phi \wedge \psi$ holds if the time-point is already labelled with both ϕ and ψ.
   - $\neg\phi$ holds if the time-point is not already labelled with ϕ.
   - $\phi \mathbf{U}_{[t_0,t_n]} \psi$ holds if:
     - *either* the time is $\leqslant t_n$ and:
       - *either* the time-point is already labelled with ψ
       - *or* it is already labelled with ϕ and the previous time point was labelled with $\phi \mathbf{U}_{[t_0,t_n]} \psi$,
     - *or* the time is $< t_0$ and the previous time point is labelled with $\phi \mathbf{U}_{[t_0,t_n]} \psi$.
   - $Q \triangleright \phi$ holds if the following procedure returns *true*:
     (a) construct the $c\pi$ process Π of this time point,
     (b) solve $Q \parallel \Pi$ and apply the algorithm to this new trace and ϕ.
4. If the initial time-point is labelled with the whole formula then return *true*, otherwise return *false*.

Note that in step 1 the context modality is treated as an atomic formula. The sub-formula of a context modality is only used by the new call of the algorithm in computing its satisfaction for the trace of the newly composed process.

4.2.1. Complexity

Now the trace is only traversed once if we are checking the fragment of $\mathcal{LBC}$ without the context modality. Therefore, the worst case for this fragment is polynomial in the size of formula $O(nf)$, a significant improvement.

If we consider the full logic, then we still see exponential worst case complexity $O(n^f)$, but improvement in some cases over the naive algorithm. In the naive algorithm we see exponential complexity in the depth of **U** nesting. In the dynamic programming algorithm we eliminate the re-computation of directly nested temporal modalities, hence polynomial time for the temporal fragment.

One immediate disadvantage of this algorithm is that, although we only traverse the trace once, we always traverse the whole trace. The recursive algorithm, working forwards along the trace, will terminate if it finds, say, a witness for $\mathbf{F}\phi$ or a counterexample to a $\mathbf{G}\phi$ before the end of the trace. Indeed, to check an atomic proposition then only the initial time point is required. This *short-circuiting* can save a lot of computation in practice.

Worse, the dynamic programming algorithm's lack of short-circuiting means that when checking a context modality the re-computation of a trace is necessarily done at every time point. This is true even, for example, when the context modality is at the top level.

With this in mind, in the next section we describe a further improved, hybrid, algorithm.

4.3. Hybrid algorithm

By combining ideas from the previous two algorithms we can devise a hybrid algorithm. We take the recursive algorithm, add the aspect of dynamic programming in a recursive manner, whilst retaining the ability to short-circuit the checking. We also make use of memoisation for optimisation.

The algorithm is based on the naive algorithm, but instead of recursing over the structure of formulae we recurse over a list (or array) of sub-formulae simultaneously. Rather, we have an outer recursion over the trace and an inner recursion over the list of sub-formulae. As in the dynamic programming algorithm, this list should be in dependency order, a post-order depth-first traversal of the formula.

To avoid recursing over sub-formulae that we have already evaluated we can memoise the result of evaluating each sub-formula at each time point, i.e. we build a lookup table mapping $\mathit{Formula} \to \mathbb{B}$ for each time point. Before evaluating a sub-formula at a time point we check if it is in the lookup table. If it is, then we do not need to re-evaluate, if not, then we evaluate and add it to the table. This is essentially what the dynamic programming approach does by labelling time points with their satisfied formulae.

We summarise the algorithm by the following functional pseudocode:

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

```
check :: [Formula] → Trace → Table
check fs []                =  emptyTable
check fs (t:ts)            =  checkSt t (check fs ts) fs

checkSt :: State → Table → [Formula] → Table
checkSt t next []          =  emptyTable
checkSt t next (f:fs)      =  insert f (eval fs t next f) (checkSt t next fs)

eval :: [Formula] → State → Table → Formula → Bool
eval fs t next (atom)      =  valid atom t
eval fs t next (φ ∧ ψ)     =  (lookup φ (checkSt t next fs))
                              AND (lookup ψ (checkSt t next fs))
eval fs t next (¬φ)        =  NOT (lookup φ (checkSt t next fs))
eval fs t next (φ U[t0,tn] ψ)  =  case time(t) of
                              < t0 : lookup (φ U[t0,tn] ψ) next
                              ⩽ tn : (lookup ψ (checkSt t next fs))
                                   OR ((lookup φ(checkSt t next fs))
                                       AND (lookup (φ U[t0,tn] ψ) next))
                              else : False
eval fs t next (Q ▷ φ)     =  check (subs φ) (solve (compose Q (proc t)))

lookup :: Formula → Table → Bool
insert :: Formula → Bool → Table → Table
subs :: Formula → [Formula]
```

The `check` function now takes a list of the sub-formulae of the formula to be checked, a trace, and returns a lookup table (`Table`) of sub-formulae which are satisfied. If the whole formula is in the returned lookup table, with a value of *true*, then it is satisfied for the trace.

The function `check` recurses over the time points and the function `checkSt` recurses over the sub-formulae, using the function `eval` determines the satisfaction of a sub-formula at a given time point. A value is retrieved from the lookup table by `lookup` and inserted by `insert`. We also have a function `subs` which gives a list of the sub-formulae of a formula. The functions `valid`, `solve`, `compose`, and `proc` are the same as in Section 4.1.

4.3.1. Complexity

For the temporal fragment we still have the same worst case complexity, but we reduce the likelihood of hitting the upper bound by exploiting short-circuiting. We evaluate over no more of the trace than is needed to return a result e.g. if we find that ϕ is true at the start of a trace and we are evaluating $\mathbf{F}\phi$ then we stop as soon as we find this witness, without evaluating over the rest of trace. This algorithm also has the advantage that we are evaluating forwards along the trace and there is no need to reverse the trace.

However, we cannot eliminate the re-computation required where nested temporal modalities are separated by a context modality. Therefore, for the full logic the hybrid algorithm can do no better than $O(n^d)$ where d is the *sandwich alternation depth*, which we define as the greatest nesting of temporal modalities separated by context modalities. This is close to, but not the same as, classic alternation depth. For example: $Q \triangleright \mathbf{G}\phi$ only requires a single trace traversal $O(n)$; so does $\mathbf{G}(Q \triangleright \phi)$ and even $Q \triangleright \mathbf{G}(Q' \triangleright \phi)$; but $\mathbf{G}(Q \triangleright \mathbf{G}\phi)$ requires multiple trace traversals $O(n^2)$ and $\mathbf{G}(Q' \triangleright \mathbf{G}(Q \triangleright \mathbf{G}\phi))$ requires $O(n^3)$.

4.4. Calls to the solver

Above, we have seen improvements in the computation time of the model-checking algorithm. However, this does disregard a major constant factor in practice. The calls made to the ODE solver by checking a context modality are computationally heavy, especially with complex models.

The key to improving the efficiency of model-checking is reducing the number of calls we have to make to the solver. One method for reducing the number of calls to the solver is to re-write the formula before model-checking, eliminating redundant context modalities. For example:

$$(Q \triangleright \phi) \wedge (Q \triangleright \psi) \mapsto Q \triangleright (\phi \wedge \psi)$$

$$Q \triangleright \left(Q' \triangleright \phi\right) \mapsto \left(Q \parallel Q'\right) \triangleright \phi$$

Each of these rules, when applied, will reduce the number of calls to the solver, required to model-check the formula, from two to one.

We also see multiple solver calls when increasing the sandwich alternation depth and formula rewrites cannot eliminate any of the calls required by sandwich alternation. Consider, for example, $\mathbf{G}(Q \triangleright \phi)$ where for every time point in the original trace we need to introduce Q and call the solver to obtain a new trace for the system composed with Q, that is $n+1$

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

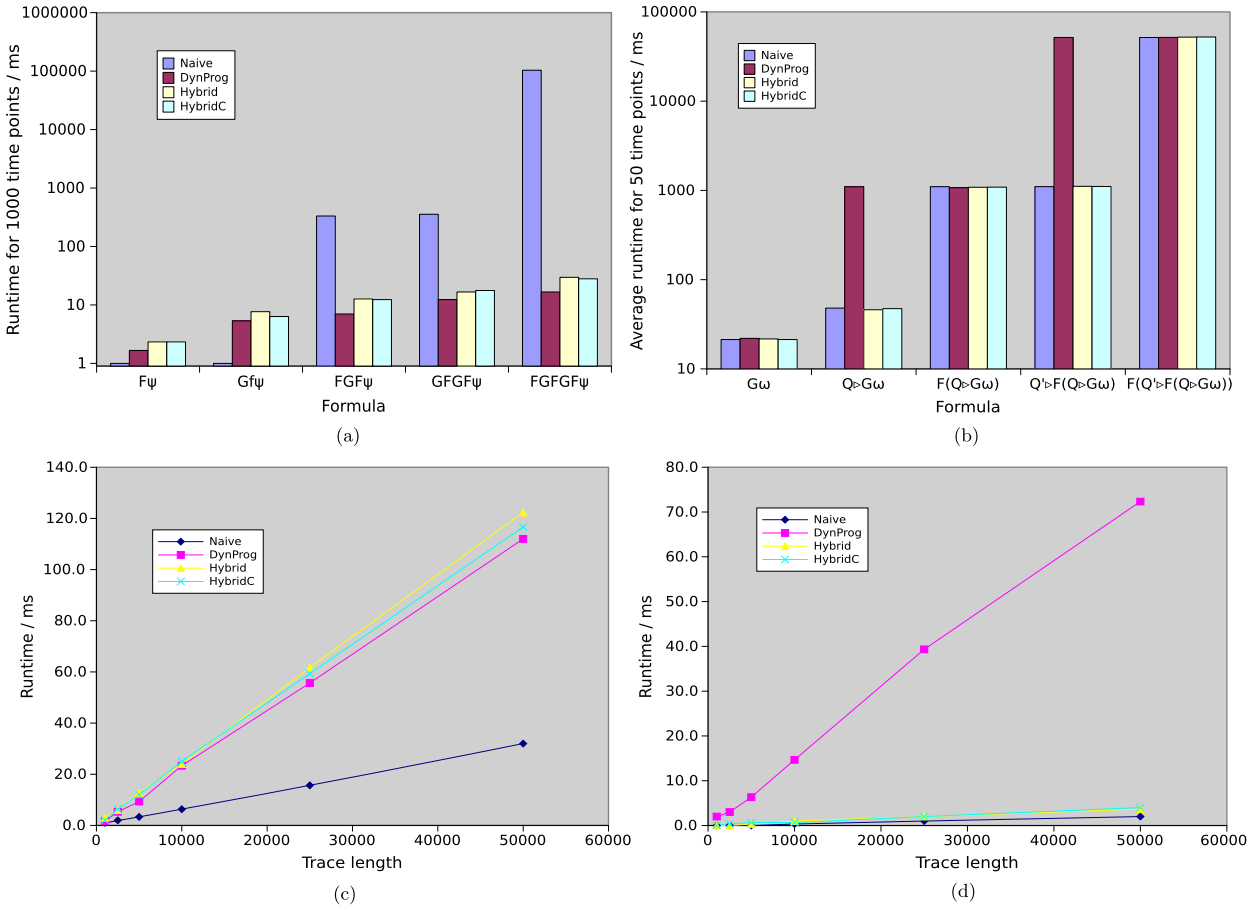

Fig. 2. Highlighted model-checker performance results: (a) runtimes for increasing temporal formula depth, (b) runtimes for increasing sandwich alternation depths of $\mathcal{LBC}$ formulae, (c) runtimes for a formula with no short-circuit potential, and (d) runtimes for a formula with short-circuit potential. In (a) and (b): ϕ is true towards the start of the trace, ψ is true only towards the end of the trace.

calls to the solver. These new calls to the solver, however, are independent and therefore can be executed concurrently. Parallelising the algorithm should give significant speedup.

4.5. Experimental results

The logic and various model-checking algorithms have been implemented as part of the Continuous π-calculus Workbench (CPiWB).[1] The CPiWB is written in Haskell and includes an interactive interpreter for cπ and $\mathcal{LBC}$ a selection of ODE solvers for producing the simulation traces required for model-checking, various outputs for analysis of cπ models, and the model-checker for $\mathcal{LBC}$.

Using the CPiWB, some experimental results were produced which support the assertions which we have made about complexity.

Throughout this section we make use of basic formulae of the following form. ϕ is a proposition which is true towards the start of the trace, ψ is a proposition which is true only towards the end of the trace, χ is a proposition which is only false towards the end of the trace, and ω is false towards the start of the trace.

Full details of the model used in the experiments, along with parameters and propositional formula values, appear in Appendix A.

4.5.1. Temporal logic fragment

First we examine the performance results of model-checking just the temporal logic fragment of $\mathcal{LBC}$. The naive algorithm has a runtime exponential in the alternation depth of the temporal modalities; Fig. 2a illustrates this. Formula depth is the greatest depth of alternating temporal modalities in the formula: e.g. $\mathbf{F}\phi$ has depth 1, $\mathbf{GF}\phi$ has depth 2, etc.; $\mathbf{G}\phi \wedge \mathbf{F}\phi$ has depth 1.

[1] http://banks.ac/software/.

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

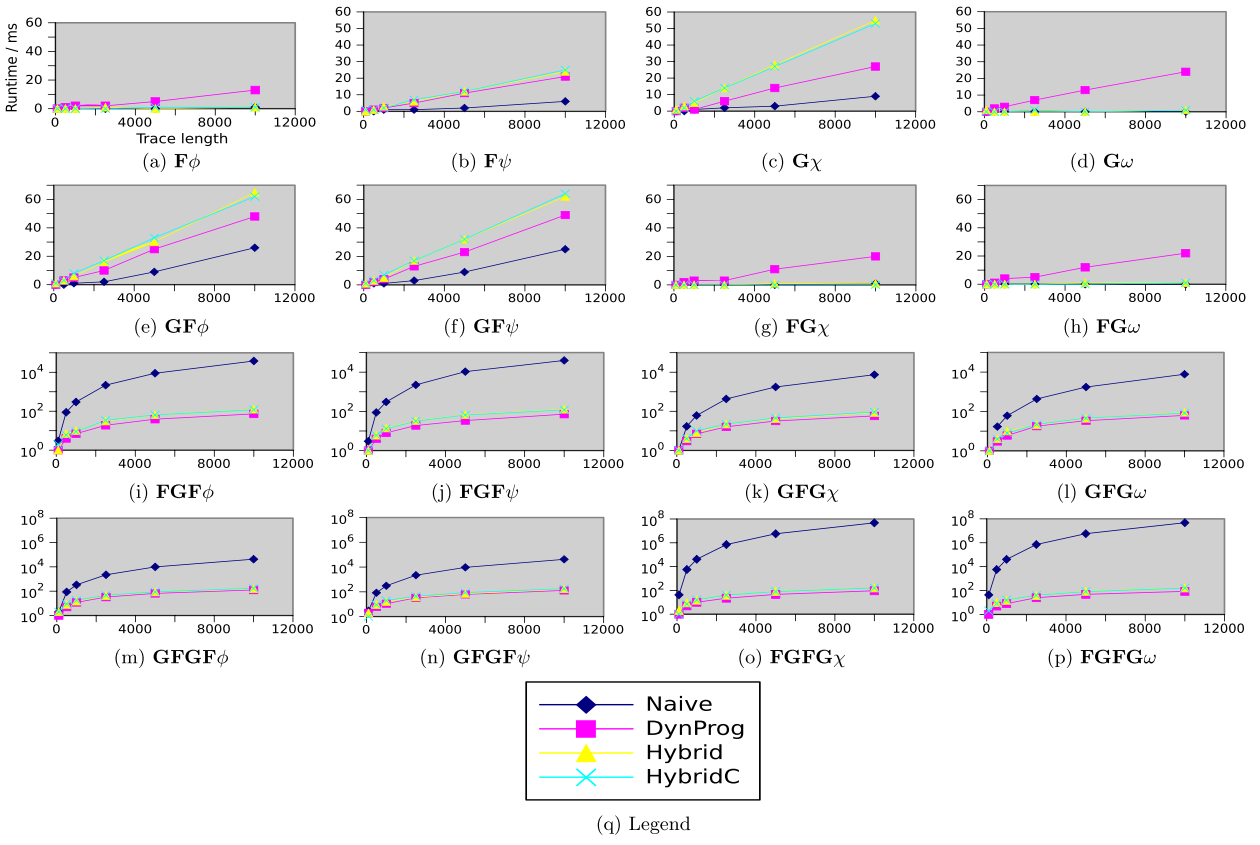

Fig. 3. Systematic performance results for the temporal fragment of $\mathcal{LBC}$. ϕ is true towards the start of the trace, ψ is true only towards the end of the trace, χ is only false towards the end of the trace, and ω is false towards the start of the trace. Note that plots (i)–(p) use a log scale for runtime.

The performance plots show runtimes for implementations of the naive algorithm, the dynamic programming algorithm, and two implementations of the hybrid algorithm. The second implementation of the hybrid algorithm (HybridC) uses a lazy circular data structure to implicitly implement the memoisation; it can be seen that this gives roughly the same performance as the more explicit implementation.

The dynamic programming algorithm does not allow the short-circuiting described in Section 4.2.1. Fig. 2c shows runtimes for checking a formula $\mathbf{F}\phi$ where ϕ is a proposition which is true towards the start of the trace. Clearly the naive algorithm, in its simplicity, has the best performance in this case and the dynamic programming and hybrid algorithms have comparable, worse performance. Fig. 2d shows runtimes for the formula $\mathbf{F}\psi$ where ψ is a proposition which is true only towards the end of the trace. Now, the naive and hybrid algorithms perform much better than the dynamic programming algorithm which does not short-circuit.

The results of a systematic exploration of runtimes for varying formulae are shown in Fig. 3.

4.5.2. $\mathcal{LBC}$

Now we examine the performance results for model-checking all of $\mathcal{LBC}$, including the context modality. Fig. 4 shows that, for most cases, the dynamic programming algorithm has a much greater runtime; its lack of short-circuiting is critical when we are forced to minimise the considerable cost of calling the ODE solver.

Fig. 4 also shows that naive algorithm has no worse runtime than the hybrid algorithm for most cases when we include the context modality. However, when we combine the context modality and nested temporal modalities, the naive algorithm has the same exponential increase in computation time incurred by nesting temporal modalities.

In Section 4.3.1 we noted that complexity increases exponentially in the *sandwich alternation depth*; Fig. 2b illustrates this. The dynamic programming algorithm increases ahead of the recursive algorithms, again, because of the lack of short-circuiting.

Fig. 5 shows that the runtime exponent does indeed correspond to the sandwich alternation depth for the hybrid algorithm. From the plots we can see that the runtimes increase with approximately the predicted exponents. For example, in Fig. 5c, running the hybrid algorithm on a formula $\mathbf{F}(Q' \triangleright \mathbf{F}(Q \triangleright \mathbf{G}\omega))$ with sandwich alternation depth 3 for increasing trace length fits the function $\ln(y) = 2.90108 \ln(x) - 0.779873$ which has a gradient of approximately 3. For all plots in Fig. 5c the correlation coefficient R^2 is greater than 0.99. We can also see that the dynamic programming algorithm increases its exponent with the addition of a context modality, rather than a context modality nested within temporal modalities.

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

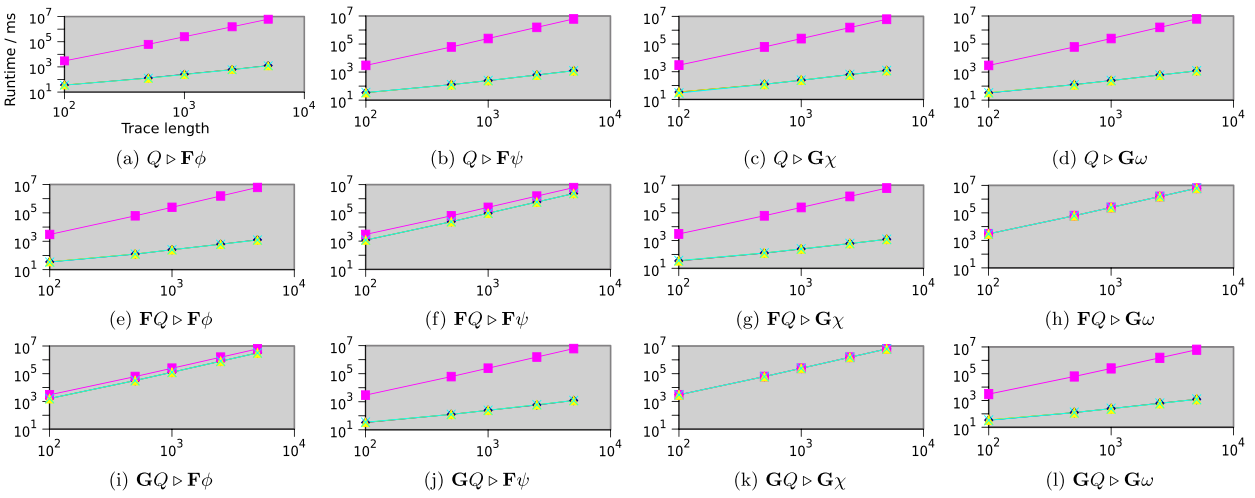

Fig. 4. Systematic performance results for the full logic $\mathcal{LBC}$. For the legend see Fig. 3q and note that these plots have a log–log scale. ϕ, ψ, χ, ω are as in Fig. 3.

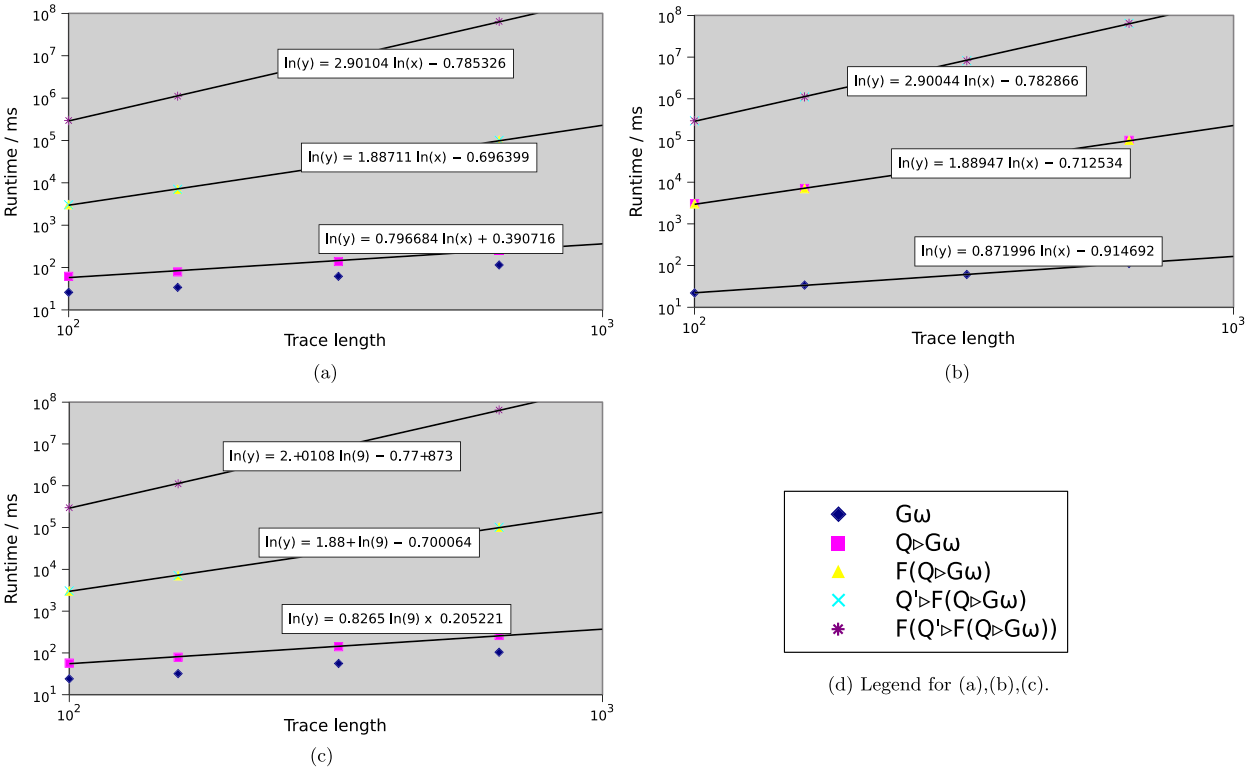

Fig. 5. Runtime exponent increase with sandwich alternation depth for (a) naive algorithm, (b) dynamic programming algorithm, and (c) hybrid algorithm.

5. Conclusion

In this paper we have introduced a logic of behaviour in context. This work represents two novel contributions.

Foremost, the logic itself is a succinct language for expressing properties of behaviour in context. Whilst the idea of the context modality comes from the guarantee operator of Cardelli and Gordon [1], the context modality has a more tractable implementation. Thus, $\mathcal{LBC}$ combined with the technique for model-checking, is a novel framework for analysing how the introduction of external influences affects the behaviour of a process.

The second contribution is the hybrid recursive algorithm for model-checking $\mathcal{LBC}$. The algorithm is a functional re-working of the algorithm of Calzone et al. [3] with the added benefit that a traversal of the trace may terminate if it is not

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

necessary to continue checking the rest of the trace. Whilst this does not improve worst case complexity, it will result in improved execution times in practice.

Some limitations of the technique have been identified. One such limitation is the time cost in practice of evaluating a context modality. We have found that the computation time can be reduced by moving to a recursive algorithm with short-circuiting. However, a naive algorithm gives an exponential increase in computation time when nesting temporal modalities.

Our hybrid algorithm negates many of the cases where the naive or dynamic programming algorithms encounter exponential growth in computation time. The hybrid algorithm, however, is still exponential in the depth of sandwich alternation — where temporal modalities are nested, but separated by context modalities.

We have also seen that the key to improving this is to minimise the number of calls we need to make to the ODE solver. Formula re-writing before model-checking provides some limited improvement, but future work in this area will investigate other methods.

Another, potentially more serious, limitation is that various assumptions are made about the precision of the simulation traces with respect to the model and loss of information. Future work in this area is likely to concentrate on improving the guarantees that can be given about the results of model-checking these continuous state-space models.

5.1. Future work

At the time of writing a case study using $\mathcal{LBC}$ is ongoing. We have built a model of Jolley et al.'s posttranslational oscillators [14] and we are conducting a number of experiments on the interactions between two or more of these systems and interactions with other systems.

Other future work can go in a number of directions. One direction is to look at extending the logic with a dual to the context modality which removes species from the system. This could be similar to the consumption operator of de Nicola and Loreti [18]. In the context of $c\pi$ this could remove some concentration of a species in the system or remove some species entirely. Consideration would need to be given to the case where a formula describes the removal of more than the concentration of a species available in the system.

The work of Calzone et al. has been generalised by Rizk et al. [20,21] to temporal logic constraint solving. The application of constraint solving and parameter search to values in the atomic propositions of $\mathcal{LBC}$ formulae would be an interesting avenue to pursue.

Another direction could be to investigate better performing model-checking algorithms. One such algorithm could come from the formula rewriting techniques of Rosu and Havelund [22]. This technique has already been used for a MITL-like logic by Bulychev et al. [23], however it remains to be determined whether the context modality can be dealt with in this scheme. It is unlikely this sort of algorithm will give an improvement in worst case time complexity, because it would perform essentially the same number of operations over the trace points and sub-formulae. However it is forward and local; i.e. only the current time point needs to be considered, no keeping of history is required.

One final, and probably the most important, direction for future development is to investigate techniques which we hope will eliminate some of the shortcomings of the trace based method. One such possible technique is the signal monitoring techniques of Maler and Nickovic [24,25]. Rather than numerically solving and checking discrete time points, the technique here is to use root-finding to detect when the atomic properties of the formula change truth value. The temporal modalities are then defined over signals representing these changes in value. Again, the challenge is to efficiently check the context modality in this setting.

Acknowledgments

We thank Jane Hillston for helpful suggestions, guidance, and proof reading; Luca Bortolussi for helpful discussions and suggestions for future work; Silvain Soliman for spotting weaknesses in our work and discussions about constraint solving. Thanks also to the anonymous reviewers for their helpful critique and suggested improvements.

This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/P50550X/1] and the Laboratory for Foundations of Computer Science.

Appendix A. Experimental reproducibility

The model used for producing the experimental results in Section 4.5 is equivalent to the model described in Section 2.3:

$$S \triangleq \left(\nu\{x \overset{r_u}{-} r, x \overset{r_r}{-} r\}\right) s\langle x\rangle.(u.S + r.P)$$

$$E \triangleq e(x).x.E$$

$$P \triangleq \tau_{r_d}.0$$

$$\Pi \triangleq c_S \cdot S \parallel c_E \cdot E \parallel c_P \cdot P$$

$$N_\Pi = \{s \overset{r_b}{-} e\}$$

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009

along with the process Q which acts and an inhibitor and is used in testing the context modality:

$$Q \triangleq c_I \cdot I \qquad I \triangleq \left(\nu\{x \overset{r_j}{-} u\}\right) i\langle x\rangle.u.I$$

$$N_Q = \{i \overset{r_i}{-} e\}$$

The parameters are as follows:

$$r_d = 0.5 \qquad c_S = 1.0$$
$$r_b = 1.0 \qquad c_E = 0.5$$
$$r_u = 0.5 \qquad c_P = 0.0$$
$$r_r = 1.0 \qquad c_I = 0.5$$
$$r_i = 2.0$$
$$r_j = 0.1$$

The propositions used in the test formulae are defined as follows:

$$\phi = [P] > 0.05 \qquad \psi = [S] \leqslant 0.01$$
$$\chi = [E] > 0.1 \qquad \omega = [E] > 0.4$$

References

[1] L. Cardelli, A. Gordon, Anytime, anywhere: Modal logics for mobile ambients, in: Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM, 2000, pp. 365–377.
[2] M. Antoniotti, A. Policriti, N. Ugel, B. Mishra, Model building and model checking for biochemical processes, Cell Biochem. Biophys. 38 (3) (2003) 271–286, http://dx.doi.org/10.1385/CBB:38:3:271.
[3] L. Calzone, N. Chabrier-Rivier, F. Fages, S. Soliman, Machine learning biochemical networks from temporal logic properties, in: Transactions on Computational Systems Biology VI, 2006, pp. 68–94.
[4] M. Kwiatkowski, A formal computational framework for the study of molecular evolution, PhD thesis, University of Edinburgh, 2010.
[5] M. Kwiatkowski, I. Stark, The continuous π-calculus: A process algebra for biochemical modelling, in: Computational Methods in Systems Biology, Springer, 2008, pp. 103–122.
[6] R. Milner, Communicating and Mobile Systems: The pi-Calculus, Cambridge University Press, 1999.
[7] A. Regev, W. Silverman, E. Shapiro, Representation and simulation of biochemical processes using the pi-calculus process algebra, in: Pacific Symposium on Biocomputing, vol. 6, World Scientific Pub. Co. Inc., 2001, pp. 459–470.
[8] A. Regev, E. Shapiro, Cells as computation, Nature 419 (2002) 343.
[9] N. Chabrier-Rivier, F. Fages, S. Soliman, The biochemical abstract machine BIOCHAM, in: Computational Methods in Systems Biology, Springer, 2005, pp. 172–191.
[10] M. Calder, S. Gilmore, J. Hillston, Automatically deriving ODEs from process algebra models of signalling pathways, in: Proceedings of Computational Methods in Systems Biology, 2005, pp. 204–215.
[11] M. Calder, A. Duguid, S. Gilmore, J. Hillston, Stronger computational modelling of signalling pathways using both continuous and discrete-state methods, in: Computational Methods in Systems Biology, Springer, 2006, pp. 63–77.
[12] C.Y. Huang, J.E. Ferrell, Ultrasensitivity in the mitogen-activated protein kinase cascade, Proc. Natl. Acad. Sci. USA 93 (19) (1996) 10078–10083.
[13] A. Clark, S. Gilmore, A. Georgoulas, J. Hillston, I. Stark, C. Banks, D. Milios, Stochastic modelling of the Kai-based circadian clock, Electron. Notes Theor. Comput. Sci. (ISSN 1571-0661) 296 (2013) 43–60, http://dx.doi.org/10.1016/j.entcs.2013.07.004.
[14] C.C. Jolley, K.L. Ode, H.R. Ueda, A design principle for a posttranslational biochemical oscillator, Cell Rep. 2 (4) (2012) 938–950, http://dx.doi.org/10.1016/j.celrep.2012.09.006.
[15] N.E. Buchler, M. Louis, Molecular titration and ultrasensitivity in regulatory networks, J. Mol. Biol. 384 (5) (2008) 1106–1119, http://dx.doi.org/10.1016/j.jmb.2008.09.079.
[16] R. Alur, T. Feder, T. Henzinger, The benefits of relaxing punctuality, J. ACM 43 (1) (1996) 116–146.
[17] L. Caires, E. Lozes, Elimination of quantifiers and undecidability in spatial logics for concurrency, Theor. Comput. Sci. 358 (2–3) (2006) 293–314.
[18] R.D. de Nicola, M. Loreti, MoMo: A modal logic for reasoning about mobility, in: Formal Methods for Components and Objects, Springer, 2005, pp. 95–119.
[19] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, U. Kummer, COPASI–a COmplex PAthway SImulator, Bioinformatics 22 (24) (2006) 3067–3074, http://dx.doi.org/10.1093/bioinformatics/btl485.
[20] A. Rizk, G. Batt, F. Fages, S. Soliman, Continuous valuations of temporal logic specifications with applications to parameter optimization and robustness measures, Theor. Comput. Sci. 412 (26) (2011) 2827–2839, http://dx.doi.org/10.1016/j.tcs.2010.05.008.
[21] F. Fages, a. Rizk, On temporal logic constraint solving for analyzing numerical data time series, Theor. Comput. Sci. 408 (1) (2008) 55–65, http://dx.doi.org/10.1016/j.tcs.2008.07.004.
[22] G. Rosu, K. Havelund, Rewriting-based techniques for runtime verification, Autom. Softw. Eng. 12 (2) (2005) 151–197.
[23] P. Bulychev, A. David, K. Larsen, A. Legay, G. Li, D.B.g. Poulsen, Monitor-based statistical model checking for weighted metric temporal logic, in: Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, Springer-Verlag, 2012, pp. 168–182.
[24] O. Maler, D. Nickovic, Monitoring temporal properties of continuous signals, in: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Springer, 2004, pp. 152–166.
[25] A. Donzé, O. Maler, Robust satisfaction of temporal logic over real-valued signals, in: Formal Modeling and Analysis of Timed Systems, Springer, 2010, pp. 92–106.

Please cite this article in press as: C.J. Banks, I. Stark, A logic of behaviour in context, Inform. and Comput. (2014), http://dx.doi.org/10.1016/j.ic.2014.01.009